Computers Math. Applic. Vol. 24, No. 3, pp. 49–59, 1992
Printed in Great Britain.

0097-4943/92 $5.00 + 0.00

THE EFFECT OF UNARY INCLUSION DEPENDENCIES ON RELATIONAL DATABASE DESIGN

YANCHUN ZHANG AND MARIA E. ORLOWSKA
Key Centre for Software Technology, Department of Computer Science
University of Queensland, Q. 4072, Australia

(Received October 1990 and in revised form November 1991)

Abstract—Functional dependencies (FDs) and inclusion dependencies (INDs) are the most fundamental database integrity constraints, and they are used in many data models. In a previous paper, we described a synthesis algorithm for the design of a relation database from FDs. In this paper, the effect of unary inclusion dependencies (UINDs) on the relational database design is studied. Though the implication problem for a set of INDs and FDs is undecidable, if attention is restricted to unary INDs, there will be a complete axiomatization and its decision problem can be solved in polynomial time. To discover new FDs and INDs from a set of FDs and UINDs, an effective algorithm is presented to find k-cycles in the multi-graph presentation of FDs and UINDs. Finally, the synthesis algorithm is enhanced by considering interaction between FDs and unary INDs.

1. INTRODUCTION

Functional dependencies (FD) and inclusion dependencies (IND) are the most fundamental database integrity constraints, and they are used in many data models. Their interaction has recently been investigated in several papers [1–3].

It is known that, when only functional dependencies are given, a third normal form (3NF) relational database can be constructed using various approaches [4–10]. The synthesis approaches described in [8,9] process a set of functional dependencies to produce a minimal annular cover, and then construct a 3NF relational database with a minimum number of relations. An expert database design system, View Creation System (VCS) described in [6,7], also collects functional dependencies from the user and applies the normalization procedure to produce the final relational database in Fourth Normal Form (4NF), in which the interaction between the system and the user is supported. But when functional dependencies and inclusion dependencies are considered simultaneously, the problem becomes more difficult. The reason is that when introducing INDs, there may be some FDs (INDs) which are implied by the set of FDs and INDs, but not implied by the FDs (INDs) alone. Hence, the relational database synthesized from old FDs may not be optimal with regard to the new FD set.

In this paper, we will extend the synthesis algorithm [8,9] for the design of relational databases by considering the interaction between FDs and unary inclusion dependencies (UINDs). In the next section, some concepts and notations of relational databases, synthesis algorithms and inclusion dependencies are reviewed. In Section 3, we consider the interaction of FDs and INDs, and restrict our attention to unary INDs. An algorithm to find k-cycles for discovering new FDs and INDs is given in Section 4. We extend the synthesis algorithm to deal with both FDs and INDs in Section 5. Conclusions are contained in Section 6.

2. CONCEPTS AND NOTATIONS

In this section, some concepts and notations on relational databases, synthesis algorithm and inclusion dependencies are briefly reviewed. For further details, we refer the reader to [3,8,11].

The authors would like to express their appreciation to the anonymous referees for their helpful comments on an earlier version of this paper.

2.1. *Functional Dependencies*

A relation schema R is a finite set of attribute names $\{A_1, A_2, \ldots, A_n\}$. Corresponding to each attribute name A_i there is a set D_i, $1 \leq i \leq n$, called the domain of A_i (DOM(A_i)). Let $D = \bigcup_{i=1}^{i=n} D_i$. A relation r on the relation schema R is a finite set of mappings $\{t_1, t_2, \ldots, t_p\}$ from R to D, with the restriction that for each mapping $t \in r$, $t(A_i)$ must be in $D_i, 1 \leq i \leq n$. These mappings are called tuples.

Functional dependencies form a family of constraints on a relation schema. A functional dependency (FD) is of the form $X \rightarrow Y$ where X and Y are sets of attributes. A relation r satisfies the FD $X \rightarrow Y$ (or $X \rightarrow Y$ holds in r) if for every two tuples in r, say u and v, $u[X] = v[X]$ implies $u[Y] = v[Y]$. X and Y are equivalent, written as $X \tilde{} Y$, if $X \rightarrow Y$ and $Y \rightarrow X$.

Let U be a set of attributes, Σ a set of dependencies. The following axioms $\mathrm{F_r}$, $\mathrm{F_a}$ and $\mathrm{F_t}$ (Armstrong's Axioms) are a complete axiomatization for the implication of FDs [4]:

$\mathrm{F_r}$ Reflexitivity: $X \rightarrow X$;
$\mathrm{F_a}$ Augmentation: $X \rightarrow Y$ implies $XZ \rightarrow YZ$;
$\mathrm{F_t}$ Transitivity : $X \rightarrow Y$ and $Y \rightarrow Z$ imply $X \rightarrow Z$.

Let F be a set of FDs for a relation $r(R)$. The *closure* of F, written F^+, is the smallest set containing F, such that Armstrong's Axioms cannot be applied to the set to yield an FD not in the set F.

Let G and F be two sets of functional dependencies. G is a cover of F (or G and F are equivalent, written as $G \equiv F$) if $G^+ = F^+$. G is nonredundant if no proper subset of it is a cover of F. If G is a nonredundant cover of F and there is no other cover containing fewer FDs than G contains, then G is a minimum cover of F.

Let R be a relation schema with attributes $A_1, A_2, \ldots, A_n$ and with the set of functional dependencies F. Let X be a nonempty subset of $\{A_1, A_2, \ldots, A_n\}$. X is a key of R if it has the following properties:

1. $x \rightarrow A_1 A_2 \ldots A_n$ is in F^+, and
2. no proper subset $Y \subset X$, such that $Y \rightarrow A_1 A_2 \ldots A_n$ is in F^+.

Let X and Y be two subsets of R, $X \rightarrow Y$ be a functional dependency. An attribute A in X or B in Y is extraneous if $(X-A) \rightarrow Y$ is in F^+ or $X \rightarrow B$ is in $((F - \{X \rightarrow Y\}) \cup \{X \rightarrow (Y - B)\})^+$. $X \rightarrow Y$ is called reduced if neither X and Y contain any extraneous attributes. If every FD $X \rightarrow Y$ in F is reduced, the set F is said to be reduced.

Let U be a set of attributes, each with an associated domain. A relation database schema $\mathbf{R}$ over U is a collection of relation schema $\{R_1, R_2, \ldots, R_p\}$, where $R_i = (S_i, K_i)$, $1 \leq i \leq p$, $\bigcup_{i=1}^{i=p} S_i = U$, k_i is the set of keys on R_i.

A relation database d on the database schema $\mathbf{R}$ (over U) is a collection of relations $\{r_1, r_2, \ldots, r_p\}$, such that for each relation schema $R(S, K)$ in $\mathbf{R}$, there is a relation r in d such that r is a relation on S that satisfies every key in K.

DEFINITION 2.1. *For a set of FDs F on R and a set $X \subseteq R$, let $E_F(X)$ be the set of FDs in F with left hand sides equivalent to X. Let $\overline{E}_F$ be a partition of F:*

$$\{E_F(X) \mid X \subseteq R \text{ and } E_F(X) \neq \emptyset\}.$$

X directly determines Y, written as $X \rightarrow. Y$ under F if we can find a nonredundant cover G for F, in which $X \rightarrow Y$ can be derived using only FDs in $G - E_G(X)$.

EXAMPLE 1. Let

$$R = \{ABCDEF\},$$
$$F = \{A \rightarrow BC,\ B \rightarrow A,\ AD \rightarrow E\} \text{ and}$$
$$G = \{AC,\ B \rightarrow A,\ BD \rightarrow E\}.$$

F and G are nonredundant and equivalent to each other.

$$E_F(A) = \{A \rightarrow BC,\ B \rightarrow A\},$$
$$E_F(AD) = \{AD \rightarrow E\}.$$

Then

$$\overline{E}_F = \{E_F(A),\ E_F(AD)\},$$
$$E_G(A) = \{A \rightarrow ABC,\ B \rightarrow A\},$$
$$E_G(AD) = \{DB \rightarrow E\},$$
$$\overline{E}_G = \{E_G(A),\ E_G(AD)\}.$$

DEFINITION 2.2. *A compound functional dependency (CFD) has the form:*

$$(X_1, X_2, \ldots, X_k) \rightarrow Y, \tag{1}$$

where $X_1, X_2, \ldots,$ and X_k are all distinct nonempty subsets of the relation schema R, and Y is also a subset of R (Y can be empty). If Y is empty set, (1) is often written as

$$(X_1, X_2, \ldots, X_k). \tag{2}$$

A relation $r(R)$ satisfies the CFD if and only if it satisfies the FDs $X_i \rightarrow X_j$ and $X_i \rightarrow Y$, for $1 \leq i, j \leq k$. The left hand side of the CFD is $(X_1, X_2, \ldots, X_k)$, X_i is a left set of the CFD, and Y the right hand side.

In other words, a CFD $(X_1, X_2, \ldots, X_k) \rightarrow Y$ is equivalent to a set of FDs $\{X_i \rightarrow X_j$ and $X_i \rightarrow Y$ for $1 \leq i, j \leq k\}$. In this sense, the concepts of covers and equivalence between F and G are still used, where F and G may be either sets of FDs, sets of CFDs, or one set of each. For example, the set of CFDs

$$G = \{(A, B),\ (AC,\ BC) \rightarrow DE\}$$

is equivalent to the set of FDs

$$F = \{A \rightarrow B,\ B \rightarrow A,\ AC \rightarrow D,\ BC \rightarrow E\}.$$

Let F be a set of CFDs containing $(X_1, X_2, \ldots, X_k) \rightarrow Y$, X_i be one of the left sets, and A be an attribute in X_i. Attribute A is shiftable if A can be moved from X_i to Y while preserving the equivalence of G.

EXAMPLE 2. Let $R = ABCDEF$, a set of CFDs

$$G = \{(AB, AC, AD) \rightarrow EF, (B) \rightarrow C, (C) \rightarrow D\}.$$

In the first CFD, the left sets AB and AC are shiftable since

$$G \equiv G_1 = \{(AD) \rightarrow ABCDEF, (B) \rightarrow C, (C) \rightarrow D\}.$$

DEFINITION 2.3. *A set of CFDs G is annular if there are no left sets, X and Z in different left sides, with $X \sim Z$ under F. An annular set G is nonredundant if no CFDs in G can be removed from G without altering the equivalence, and no CFD in G contains a shiftable left set. Otherwise, G is redundant.*

DEFINITION 2.4. *Let G be a nonredundant annular set. A CFD $(X_1, X_2, \ldots, X_k) \rightarrow Y$ in G is reduced if no left set contains a shiftable attribute and the right side contains no extraneous attributes.*

EXAMPLE 3. In Example 2, G is an annular set since the equivalent left sets AB, AC and AD are in the same left side, and B, C and AB are not equivalent. G is redundant since it contains

shiftable left sets AB and AC in the first CFD. After the shifting operation, G_1 is nonredundant. But G_1 is not reduced, since it contains extraneous attributes in the right side of the first CFD. $G_2 = \{AD \rightarrow BEF, B \rightarrow C, C \rightarrow D\}$ is a reduced annular cover of G.

DEFINITION 2.5. *Let G be a nonredundant annular set. G is minimum if there is no other nonredundant annular cover of G containing fewer left sets than G contains. If every CFD is reduced, then G is a reduced minimum annular set.*

In Examples 2 and 3, both G_2 and G_1 are minimum annular covers of G, but only G_2 is a reduced minimum annular cover.

In order to find a minimum annular cover for a set G of FDs, we first find a reduced minimum cover F for G, then combine FDs with equivalent left sides into single CFDs. After this conversion, we obtain a minimum annular cover F_1 of G. F_1 is not necessarily reduced, so a reduction step is still needed to get a reduced minimum annular cover.

2.2. Description of the Synthesis Algorithm SYNTHESIZER

The synthesis algorithm in [8,9] is based on finding a minimum cover. Given a relation schema R and a set of F of FDs over R, we follow the synthesis steps below:

(1) Find a nonredundant cover F_1 of F.
(2) Find a minimum cover F_2 of F_1.
(3) Find a reduced minimum cover F_3 of F_2.
(4) According to $\overline{E}_{F_3}$ (Definition 2.1), construct a nonredundant annular cover F_4 of F_3 by combining FDs with equivalent left sides into single CFDs.
(5) Find a reduced minimum annular cover F_5 of F_4 by shifting and reducing.
(6) For each CFD $(X_1, X_2, \ldots, X_k) \rightarrow Y$ in F_5, construct a relation schema:

$$R = \{X_1X_2 \ldots X_kY\}, \text{ with designed key } K = \{X_1, X_2, \ldots, X_k\}.$$

(7) Return the set of relation schema R constructed in Step 6.

The synthesis procedure above can be implemented automatically by a series of algorithms (NONREDUN, REDUCE, MINIMIZE, SYNTHESIZER and so on) in $O(n^2)$ time, on inputs of length n [8]. The final relational database schema R has the following properties [8]:

(1) F is complete characterized by R, that is:

$$F \equiv \{K_i \rightarrow R_i \mid R_i \text{ is in } R \text{ and} K_i \text{ is a designated key of } R_i\}.$$

(2) Every relation schema R_i in **R** is in 3NF with respect to F,
(3) There is no database schema with fewer relation schemas than **R** satisfying F and the Properties (1) and (2).

EXAMPLE 4. Given a set U of attributes and a set F of FDs, $U = ABCDEJGHI$,

$$F = \{BCD \rightarrow J, C \rightarrow DI, DI \rightarrow C, G \rightarrow A, BD \rightarrow G, BC \rightarrow E, J \rightarrow EHC, E \rightarrow HJB, BDI \rightarrow A\}.$$

Now we follow the synthesis procedure below:

(1) Find a nonredundant cover. Delete a redundant FD $BCD \rightarrow J$, a nonredundant cover $F_1 = F - \{BCD \rightarrow J\}$;
(2) Find a minimum cover. Since $BC \equiv BDI$ and $BDI \rightarrow .\ BC$ (BDI directly determines BC), change $BDI \rightarrow A$ to $BC \rightarrow A$, then combine the FDs with same left side into one FD, a minimum cover

$$F_2 = \{C \rightarrow DI, DI \rightarrow C, G \rightarrow A, BD \rightarrow G, BC \rightarrow EA, J \rightarrow EHC, E \rightarrow HJB\};$$

(3) Find a reduced minimum cover. F_2 is also a reduced minimal cover;

(4) Construct a nonredundant annular cover. Since $\overline{E}_{F_2} = \{\{C \to DI, DI \to C\}, \{G \to A\}, \{J \to EHC, E \to HJB, BC \to EA\}, \{BD \to G\}\}$, combine FDs with equivalent left sides into single CFDs, a minimum annular cover

$$F_4 = \{(C, DI), (BD) \to G, (G) \to A, (E, J, BC) \to HA\};$$

(5) Find a reduced minimum annular cover. F_4 is not reduced, since attribute A in the last CFD of F_4 is extraneous. Remove A from the CFD, obtain a reduced annular cover

$$F_5 = \{(C, DI), BD \to G, G \to A, (E, J, BC) \to H\};$$

(6) Construct the final data base schema $D_b = \{R_1, R_2, R_3, R_4\}$, where

$$R_1 = (CDI, \{C, DI\}),$$
$$R_2 = (BDG, \{BD\}),$$
$$R_3 = (AG, \{G\}),$$
$$R_4 = (BCEJH, \{E, J, BC\}).$$

2.3. Inclusion Dependencies

Functional dependencies are certainly the most important and widely studied integrity constraints for relational databases. Another important integrity constraint is the inclusion dependency (IND). In the following paragraphs, we review some concepts of inclusion dependencies.

DEFINITION 2.6. *An Inclusion Dependency is a statement of the form* $R.A_1A_2 \dots A_m \subseteq S.B_1B_2 \dots B_m$, *where* A_i *and* B_i *are attributes of two relations named* R *and* S. *The inclusion dependencies hold for a database if for each tuple* $t \in r$, *there is a tuple* $t' \in s$ *with* $t[A_i] = t'[B_i], i = 1, \dots, m$. *If* $m = 1$, *we have a unary IND.*

For the implication problem of INDs, the following inference rules are sound and complete [3]:

$\mathrm{ID_r}$ Reflexivity: $R[X] \subseteq R[X]$.

$\mathrm{ID_p}$ Projection and Permutation: If $R[A_1A_2 \dots A_m] \subseteq S[B_1B_2 \dots B_m]$, then $R[A_{i_1}, \dots, A_{i_k}] \subseteq S[B_{i_1} \dots B_{i_k}]$, for each sequence $i_1, i_2, \dots, i_k$ of distinct integers from $\{1, \dots, m\}$.

$\mathrm{ID_t}$ Transitivity: If $R[X] \subseteq S[Y]$ and $S[Y] \subseteq T[Z]$, then $R[X] \subseteq T[Z]$.

A decision procedure is adopted for the problem of INDs. (For example, for determining if $\Sigma \Rightarrow \sigma$, where Σ is a set of INDs and σ is a single IND. Say that σ is the IND $R_A[A_1A_2 \dots A_m] \subseteq R_B[B_1B_2 \dots B_m]$.)

(1) Initialize set Z by letting it contain the single expression $R_A[A_1A_2 \dots A_m]$.

(2) If Z contains an expression $S[X]$, and if an IND $S[X] \subseteq T[Y]$ can be obtained from a member of Σ by $\mathrm{ID_p}$ (projection and permutation), then add $T[Y]$ to the set Z.

(3) Apply Step (2) repeatedly, until either $R_B[B_1B_2 \dots B_m]$ appears in Z or until it is no longer possible to add an expression to Z by using Step (2), whichever comes first.

(4) $\Sigma \Rightarrow \sigma$ if and only if $R_B[B_1B_2 \dots B_m]$ is in the resulting set Z.

This decision procedure is nondeterminiinistic, since we do not specify the order in which the INDs in Σ are applied to members of Z in Step (2). The procedure is quite similar to a decision procedure for FDs. However, there is a major difference. The FD decision procedure's time complexity is linear. Unfortunately, however, in the case of INDs, the procedure requires superpolynomial time [1].

3. IMPLICATION OF FDS AND INDS

When considering the FDs and INDs together, there may be some FDs (INDs) which are implied by the set of FDs and INDs, but not implied by the FDs (INDs) alone, i.e., the following two situations may occur:

(a) a set of FDs and INDs imply a new FD, and

(b) a set of FDs and INDs imply a new IND.

EXAMPLE 5.

(1) From $R : CD \subseteq AB$ and $S : A \rightarrow B$ derive $R : C \rightarrow D$,
(2) From $R[CD] \subseteq S[AB]$, $S : A \rightarrow B$ and $R[CF_1 \ldots F_k] \subseteq S[AE_1 \ldots E_k]$, derive $R[CDF_1 \ldots F_k] \subseteq S[ABE_1 \ldots E_k]$.

Though inclusion dependencies have a simple complete axiomatization, just as FDs do, when FDs and INDs are considered together, there is no k-ary complete axiomatization. The implication problem for a set of INDs and FDs is undecidable. From the point of view of practical database design, INDs are very important, but the decision problem for INDs is computationally hard. If we restrict the attention to unary inclusion dependencies, then the decision problem is solvable in polynomial time.

THEOREM 2. *The following axiomatization is complete for UIND + FD finite implication [1]:*

(1) *Axiom schemes* (F_r), (ID_r), (F_a), (ID_p), (F_t), (ID_t), *from Section 2; and*
(2) *A cycle rule: For each odd positive integer k and attributes $C_0, C_1, \ldots, C_k$, we have a cycle rule* (Ck):

$$\text{From } C_0 \rightarrow C_1 \text{ and } C_1 \supseteq C_2 \ldots C_{k-1} \rightarrow C_k \text{ and } C_k \supseteq C_0,$$
$$\text{derive } C_1 \rightarrow C_0 \text{ and } C_2 \supseteq C_1 \ldots C_k \rightarrow C_{k-1} \text{ and } C_0 \supseteq C_k.$$

An FD/I-path is a sequence of attributes $A_1, A_2, \ldots, A_k$, such that $A_1 \rightarrow A_2, A_3 \rightarrow A_4, \ldots$ and $A_2 \supseteq A_3$, $A_4 \supseteq A_5, \ldots$; the length of the FD/I-path is termed k. A simple FD/I-path contains each attribute at most once in its specification. A simple cycle is a simple path with the exception that the first and last attributes are the same and k is an odd number. For simplicity, we refer to FD/I-path, simple FD/I-path, and FD/I-cycle as path, simple path and k-cycle, respectively.

The multi-graph for FDs and INDs is now introduced.

DEFINITION 3.1. *Let U be a set of attributes, $\Sigma_I = \{\sigma_1, \sigma_2, \ldots, \sigma_q\}$ a set of INDs, and $\Sigma_F = \{f_1, f_2, \ldots, f_p\}$ a set of FDs. We shall represent them as a multigraph, denoted by $G_\Sigma = (V, E)$, where V is a nonempty finite set of nodes and E is a nonempty set of arcs, such that*

(1) *For every attribute A in U, there is a simple node labeled A in V;*
(2) *For every FD $X \rightarrow Y$ in F, with $|X| > 1$, there is a compound node labeled X in V;*
(3) *For every FD $X \rightarrow Y$ in F, where $Y = A_1 A_2 \ldots A_k$, there are full arcs from the node X to the nodes labeled $A_1, A_2, \ldots, A_k$ in E; and*
(4) *For every compound node X in V, labeled $A_1 A_2 \ldots A_k$, there are dotted arcs from node X to all simple nodes (component nodes of X) labeled $A_1, A_2, \ldots, A_k$ in E.*
(5) *There is a double arrow arc from A to B if $B \subseteq A$.*

The set of simple (compound) nodes is denoted V^s (V^c, respectively), and $V = V^s \cup V^c$. The set of full arcs (dotted arcs) is denoted E^f (E^d, respectively), the set of double arrow arcs is denoted by E^I, and $E = E^f \cup E^d \cup E^I$.

EXAMPLE 6. Given a set of FDs $F = \{A \rightarrow FBC, C \rightarrow D, EBD \rightarrow H, BD \rightarrow E\}$, and a set of INDs $I = \{D \supseteq B, E \supseteq H\}$, the corresponding multi-graph $GF = \langle V, E \rangle$ is given in Figure 1.

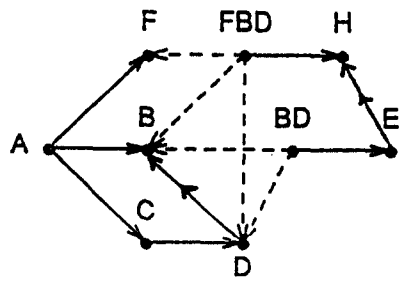

Figure 1. Example of a multi-graph.

EXAMPLE 7. One can think of the formal rules of UINDs + FDs as operations on G_Σ. Consider in particular the k-cycle rule for finite implications of FDs and unary INDs. For an odd positive integer $k = 3$ and attributes C_0, C_1, C_2, C_3, we have a cycle rule (C3):

$$\text{From } C_0 \rightarrow C_1 \text{ and } C_1 \supseteq C_2,\ C_2 \rightarrow C_3 \text{ and } C_3 \supseteq C_0,$$
$$\text{derive } C_1 \rightarrow C_0 \text{ and } C_2 \supseteq C_1 \ldots C_3 \rightarrow C_2 \text{ and } C_0 \supseteq C_3.$$

Figure 2 depicts the cycle rule for $k = 3$. When an alternative cycle like the one on the left exists in $G(\Sigma)$, add to $G(\Sigma)$ the reverse cycle (relation names have been omitted here).

It should be noted that, when using the k-cycle rules, the k-cycle

$$C_0 \rightarrow C_1 \text{ and } C_1 \supseteq C_2,\ C_2 \rightarrow C_3 \text{ and } C_3 \supseteq C_0$$

may not be contained in the multi-graph, which means that some of the FDs (or INDs) may not be in the multi-graph, but are implied or derived from the FDs (INDs) in the multi-graph. So, when we consider k-cycles, they are not only those contained in the multi-graph, but also those implied k-cycles. A k-cycle is termed "implied" as long as there are some FDs (or INDs) of the k-cycle not included in the multi-graph, but derived from other FDs in the graph.

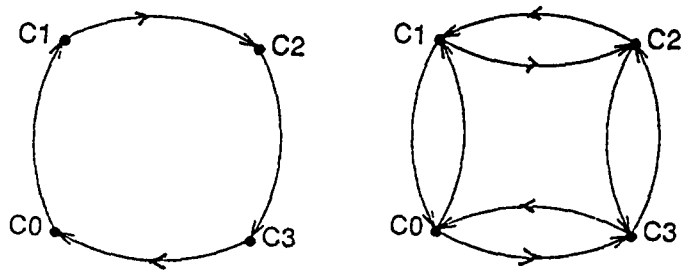

Figure 2. A 3-cycle rule.

To discover the new FDs and INDs, we need to find all such simple k-cycles. In the next section, we present an efficient algorithm to find all these k-cycles (either implied or not) from a multi-graph.

4. AN EFFICIENT ALGORITHM TO FIND SIMPLE k-CYCLES IN A MULTI-GRAPH.

In the literature, there are many efficient algorithms for finding all simple cycles in a graph and/or a directed graph [12,13]. For example, the Algorithm EC [12] has the complexity of $O(V + E + E * N)$, where N is the number of cycles listed, V is the number of vertices, E is the number of arcs of the directed graph. Our algorithm below finds all k-cycles from the multi-graph, but has the same complexity as that of EC.

The algorithm is named KC for "k-cycle." The functional blocks of KC are denoted by KC_1, KC_2, etc. KC requires that there be assigned to the vertices, in any order, the integer designators 1, 2, ..., N. The algorithm uses four principle arrays in addition to the array describing the graph. The first is a 1-dimensional array, P, containing the vertices of a simple path. The second is a 2-dimensional array, H, and is initially zeroed, and used for backtracking. The third and fourth are 2-dimensional arrays, A and B, used for storing the closure of vertex regarding FDs and UINDs, respectively. Simple path building in P is the basic process of KC. The first path is started as vertex 1. A path is extended from its end, one arc at a time, with three conditions checked before a tentative extension is performed:

(1) The extension vertex cannot be in P.
(2) The extension vertex value must be larger than that of the first vertex of P.
(3) The extension vertex cannot be closed to the last vertex in P. H contains the list of vertices closed to each vertex.

Condition (1) assures that a simple path is being formed. Condition (2) assures that each cycle will only be considered once. The search for a particular cycle will always start from its lowest valued integer (node). Condition (3) assures that no simple path is considered more than once. Therefore, the following results hold:

(1) KC is finite and, therefore, indeed an algorithm;
(2) Every k-cycle will be printed by KC;
(3) No simple k-cycle will be printed twice by KC.

Because of the three results above, the algorithm is effective.

KC begins by specifying the multi-graph G. Each vertex is given a number, $V = \{1, 2, \ldots, N\}$. The output of KC is a listing of k-cycles. Each cycle is given in vertex sequence form, $[v_1, v_2, \ldots, v_k], k \leq N$. The initial vertex is not repeated as the terminal vertex, since this is redundant. The cycles are outputed as they are found.

In the presentation of KC below, each significant operation is given a numerical label for ease of reference. The action flows one block to the next unless a "go to" statement is encountered. The example of Figure 3 can be used to help follow the algorithm.

Algorithm KC

KC1: [Initialize]

1: Read N and G.
$P \leftarrow 0$
$H \leftarrow 0$
$k \leftarrow 1$
$P[1] \leftarrow 1$
$t \leftarrow 0$
for $i = 1, \ldots, N$
$A[i] \leftarrow i^+{}_F$
$B[i] \leftarrow i^+{}_{IND}$

KC2: [Path Extension]

2: Search $A[P[k]]$ or $B[P[k]]$.
if odd $(t + k)$, search $A[P[k]]$;
otherwise search $B[P[k]]$, for $j = 1, 2, \ldots, N$
such that the following three conditions are satisfied,
(1) $j > P[1]$
(2) $j \in P$
(3) $j \in H[P[k]]$

3: if this j is found, extend the path,
$k \leftarrow k + 1$
$P[k] \leftarrow j$
goto KC2

4: if no j meets the above conditions, the path cannot be extended.

KC3: [Cycle Confirmation]

5: if odd (k), no cycle, goto KC4
6: if $t = 0$ & $P[1] \in A[k]$, then no cycle has been formed, goto KC4.
7: if $t = 1$ & $P[1] \in B[k]$, then no cycle has been formed, goto KC4.
8: otherwise a cycle is reported,
Print P.

KC4 [Vertex Closure]

9: if $k = 1$, then all of the cycle containing vertex $P[1]$ have been considered.
Goto KC5.

10: otherwise, $H[P[k], m] \leftarrow 0, m = 1, \ldots, n$
for m such that $H[P[k-1], m]$ is the leftmost zero in the P[k-1] row of H,
$H[P[k-1], m] \leftarrow P[k]$
$P[k] \leftarrow 0$

$k \leftarrow k - 1$
Goto KC2

KC5: [Advanced Initial Vertex]

11: if $P[1] = N$ then,
Goto KC6

12: otherwise,
$P[1] \leftarrow P[1] + 1$
$k \leftarrow 1$
$H \leftarrow 0$
Goto KC2

KC6: [Terminate]

13: if $t = 0$, then $t \leftarrow 1$
$P \leftarrow 0$,
$H \leftarrow 0$,
$k \leftarrow 1$,
$P[1] \leftarrow 1$.
Goto KC2

14: otherwise terminate.

EXAMPLE 8. Let us evaluate the k-cycles of the multi-graph in Figure 3 as an example of the application of KC. Table 1 presents the values of P and actions taken as the algorithm progresses. When the algorithm terminates, two cycles, C1: $\langle 1, 2, 4, 5 \rangle$ and C2: $\langle 1, 5 \rangle$, are obtained.

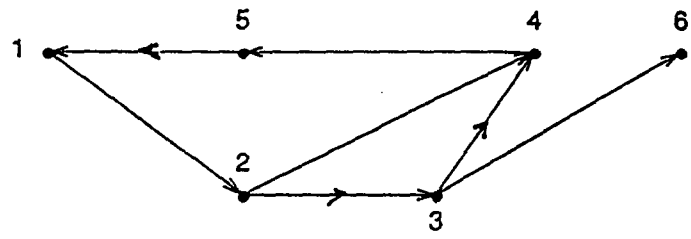

Figure 3. A multi-graph.

5. THE ENHANCED SYNTHESIS ALGORITHM

Given a set of FDs, one can synthesize the FDs to construct a 3NF relation database. However, introducing the UINDs, there may be some new FDs (not implied by the set of FDs) which hold, in which case the structure of the relational database should be constructed to fit the new FDs. To improve the quality of the RDB, the synthesis algorithm is now enhanced by using cycle rules for adding new FDs and then synthesizing the FDs.

Enhanced Synthesis Algorithm

Input: A set of FDs Σ_F and a set of UINDs Σ_I.
Output: 3NF relations.
Step 1: Add new FDs to Σ_F, adding new UINDs to Σ_I by using cycle rules (by Algorithm KC).
Step 2: Find a reduced minimum annular cover Σ_{F_m} of new Σ_F with respect to FDs.
Step 3: Find a nonredundant cover Σ_{I_m} of new Σ_I with respect to UINDs.
Step 4: Construct relations from the minimal annular cover obtained at Step 3, and add inclusion constraints to the related relation schemas.

Table 1. Evaluation of k-cycles of the graph in Figure 3.

P	Actions on attained path
1 0 0 0 0 0	
1 2 0 0 0 0	
1 2 3 0 0 0	
1 2 3 6 0 0	no cycle, $H(3,1) \leftarrow 6$
1 2 3 0 0 0	no cycle, $H(2,1) \leftarrow 3$
1 2 0 0 0 0	
1 2 4 0 0 0	
1 2 4 5 0 0	print cycle, $H(4,1) \leftarrow 5$
1 2 4 0 0 0	no cycle, $H(4,1) \leftarrow 0, H(2,2) \leftarrow 4$
1 2 0 0 0 0	no cycle, $H(2,1) \leftarrow 0, H(1,1) \leftarrow 2$
1 0 0 0 0 0	
1 5 0 0 0 0	print cycle, $H(1,2) \leftarrow 5$
1 0 0 0 0 0	no cycle, advance $P[1]$, clear H

P	Actions on attained path
2 0 0 0 0 0	
2 4 0 0 0 0	no cycle, $H(2,1) \leftarrow 4$
2 0 0 0 0 0	
2 3 0 0 0 0	
2 3 6 0 0 0	no cycle, $H(3,1) \leftarrow 6$
2 3 0 0 0 0	no cycle, $H(2,2) \leftarrow 3$
2 0 0 0 0 0	no cycle, advance $P[1]$, clear H
3 0 0 0 0 0	
3 4 0 0 0 0	
3 4 5 0 0 0	no cycle, $H(4,1) \leftarrow 5$
3 4 0 0 0 0	no cycle, $H(4,1) \leftarrow 0$, $H(3,1) \leftarrow 4$
3 0 0 0 0 0	no cycle, advance $P[1]$, clear H
4 0 0 0 0 0	
4 5 0 0 0 0	no cycle, $H(4,1) \leftarrow 5$
4 0 0 0 0 0	no cycle, advance $P[1]$, clear H
5 0 0 0 0 0	no cycle, advance $P[1]$
6 0 0 0 0 0	no cycle, terminates
⟨ 1 2 4 5 ⟩	cycles reported
⟨ 1 5 ⟩	

To generate the new FDs by the cycle rules, we only need to find all the simple k-cycles by Algorithm KC from the multi-graph. Once all such cycles are found, we can put their reverse cycles to the multi-graph, or put new dependencies in Σ if they are not implied by original FDs (or INDs, respectively). Finally, following the synthesis procedure, we can construct the relational database with inclusion dependencies attached as integrity constraints.

EXAMPLE 9. Suppose a Universe of Discourse is described by a set of FDs, $F = \{A \rightarrow B, C \rightarrow D, B \rightarrow E\}$ and a set of UINDs, $I = \{B \supseteq C, D \supseteq A\}$.

Since F is a minimal cover of itself, using the traditional synthesis algorithm, one would obtain the relational database schema:

$$R_1 = (AB, \{A\})\ R_1.A \supseteq R_3.D,$$
$$R_2 = (BE, \{B\}),$$
$$R_3 = (CD, \{C\})\ R_3.C \supseteq R_2.B.$$

When using the enhanced synthesis algorithm, we can discover the new FDs $B \rightarrow A$, $D \rightarrow C$, and UINDs $C \supseteq B, A \supseteq D$, by the cycle rule. From the new set of FDs, we can produce the following database schema:

$$R_{12} = (ABE, \{A, B\})\ R_{12}.B \subseteq R_{3'}.C,\ R_{12}.A \subseteq R_{3'}..D$$
$$R_{3'} = (CD, \{C, D\})\ R_{3'}.C \subseteq R_{12}.B, R_{3'}.D \subseteq R_{12}.A$$

Obviously, the latter schema is preferred, since it contains fewer relation schemas and carries the same semantics.

6. CONCLUSIONS

In this paper, the effect of unary inclusion dependencies on the relational database design is studied. Though the implication problem for a set of INDs and FDs is undecidable, if we restrict

attention to unary INDs, there will be a complete axiomatization and its decision problem can be solved in polynomial time. To discover new FDs and INDs from a set of FDs and UINDs, we present an effective algorithm to find k-cycles from the multi-graph presentation of FDs and UINDs. Finally, we enhanced the synthesis algorithm by considering the interaction betweeen FDs and unary INDs.

References

1. M.A. Casanova, R. Fagin and C.H. Papadimitriou, Inclusion dependencies and their interaction with functional dependencies, *J. Computer and System Science* **28** (1), 29–59 (1984).
2. A.K. Chandra and M.Y. Vardi, The implication problem for functional and inclusion dependencies is undecidable, *SIAM J. Comput.* **14** (3), 671–677 (1985).
3. S.S. Cosmdakis, P.C. Kanellakis and M.Y. Vardi, Polynomial-time implication problems for unary inclusion dependencies, *J. ACM* **37** (1), 15–46 (1990).
4. P.A. Bernstein, Synthesising third normal form relations from functional dependencies, *ACM Trans. on Database Design Systems* **1** (4), 277–297 (1976).
5. G. Ausillo, Graph algorithms for functional dependencies manipulation, *JACM* **30** (4), 752–766 (1984).
6. V. Storey and R. Goldstein, A methodology for creating user views in database design, *ACM TODS* **13** (3), 305–338 (1988).
7. V. Storey and R. Goldstein, Design and development of an expert database system, *Int. J. of Expert System* **3** (1), 31–63 (1990).
8. Y. Zhang and M.E. Orlowska, Some properties of synthesis method for the design of relational databases, In *Proceedings of International Symposium for Young Computer Professionals*, Beijing, pp. 197–202, (1989).
9. Y. Zhang and M.E. Orlowska, An improvement on the automated tool for relational database design, *Inform. System* **15** (6), 647–651 (1990).
10. Y. Zhang and M.E. Orlowska, A new polynomial time algorithm for relational database design, *Inform. System* **17** (2) (1992).
11. D. Maier, *Theory of Relational Databases*, Computer Science Press, Rockville, MD, U.S.A., (1983).
12. J.C. Tiernan, An efficient algorithm to find the elementary circuits of a graph, *Communication of ACM* **13** (12), 722–726 (1970).
13. H. Weinblatt, A new search algorithm for finding the simple cycles of a finite directed graph, *JACM* **19** (1), 43–56 (1972).